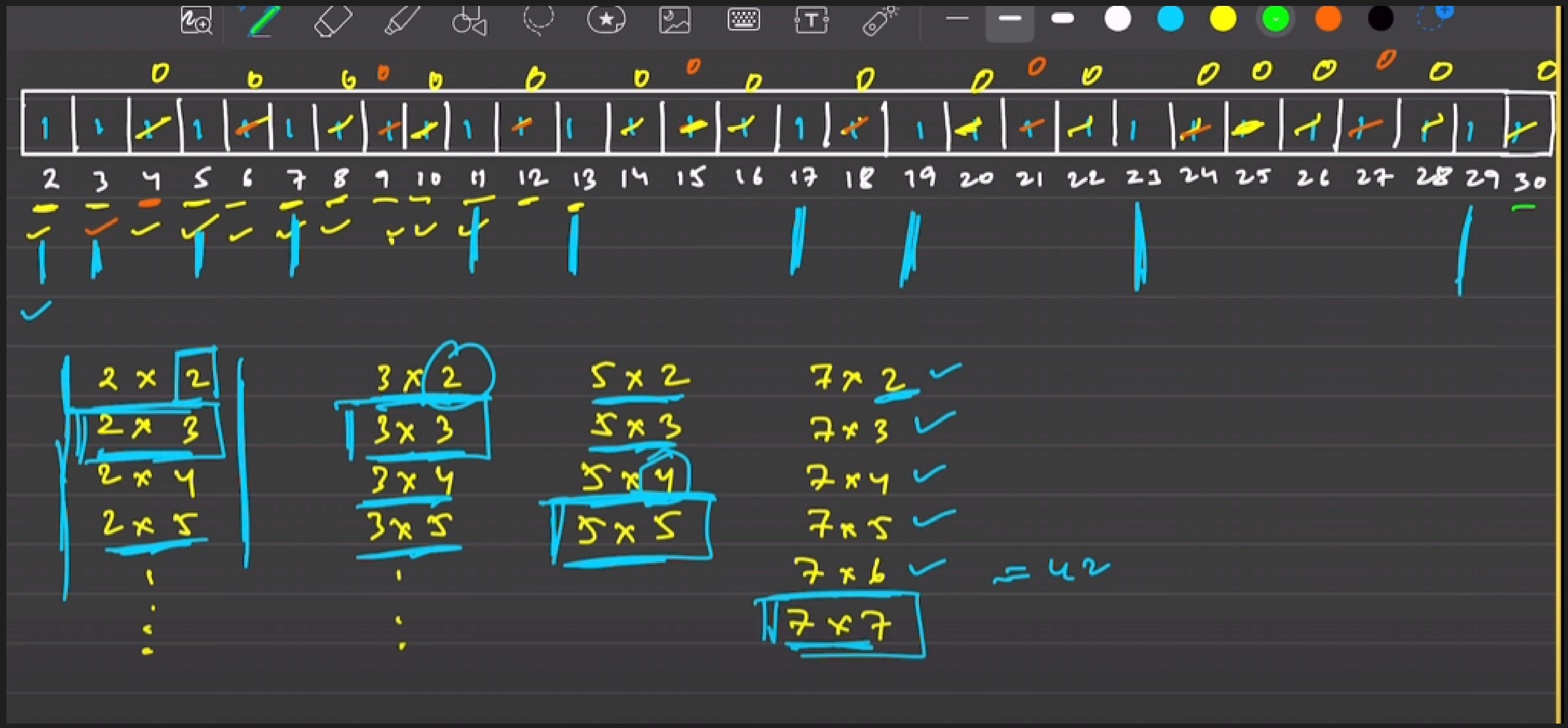

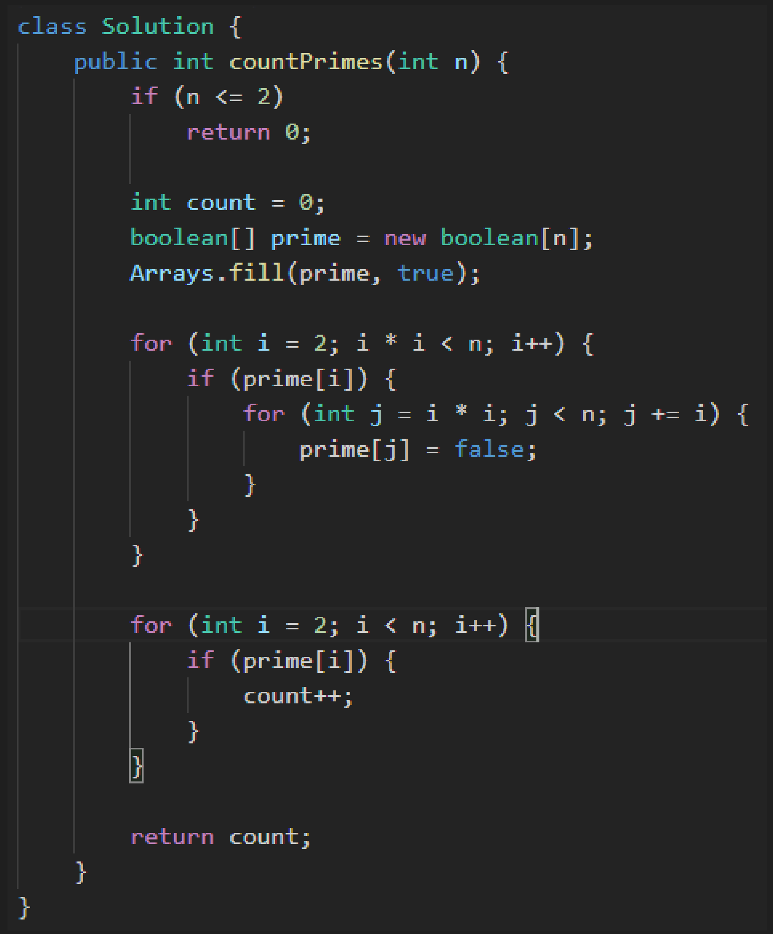

```java
class Solution {
    public int countPrimes(int n) {
        if (n <= 2)
            return 0;

        int count = 0;
        boolean[] prime = new boolean[n];
        Arrays.fill(prime, true);

        for (int i = 2; i * i < n; i++) {
            if (prime[i]) {
                for (int j = i * i; j < n; j += i) {
                    prime[j] = false;
                }
            }
        }

        for (int i = 2; i < n; i++) {
            if (prime[i]) {
                count++;
            }
        }

        return count;
    }
}
```

For Sieve of Eratosthenes printing primes till n

One key observation is that we can start inner loop from i * i because all the previous elements are already marked(striver video)

Also since we are already calculating the multipliers for number in internal loop we can limit external loop to sq root of n.

time complexity is O (log (log n)) for main algo and O(n) for filling prime array and counting primes.
TC - O(n) + O(n log (log N)) + O(n)

space complexity is O(n) for array